Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Avaliação de Qualidade de Software Baseada em Métricas Estáticas - Um Estudo de Caso no Tribunal de Contas da União

Lucas Neto Moreira

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientador
Prof. Dr. Rodrigo Bonifácio

Brasília
2015

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Coordenador: Prof. Dr. Homero Luiz Piccolo

Banca examinadora composta por:

Prof. Dr. Rodrigo Bonifácio (Orientador) — CIC/UnB
Prof. Dr. George Marsicano — CIC/UnB
Marcelo da Silva Sousa — TCU

CIP — Catalogação Internacional na Publicação

Moreira, Lucas Neto.

Avaliação de Qualidade de Software Baseada em Métricas Estáticas - Um Estudo de Caso no Tribunal de Contas da União / Lucas Neto Moreira. Brasília : UnB, 2015.
83 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2015.

1. Qualidade, 2. Métricas, 3. Inspeção Contínua

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Avaliação de Qualidade de Software Baseada em Métricas Estáticas - Um Estudo de Caso no Tribunal de Contas da União

Lucas Neto Moreira

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof. Dr. Rodrigo Bonifácio (Orientador)
CIC/UnB

Prof. Dr. George Marsicano
CIC/UnB

Marcelo da Silva Sousa
TCU

Prof. Dr. Homero Luiz Piccolo
Coordenador do Bacharelado em Ciência da Computação

Brasília, 13 de Março de 2015

Dedicatória

Dedico esta monografia à minha família, e à minha namorada, Bruna, que foram de vital importância para a realização deste trabalho.

Agradecimentos

Agradeço profundamente aos meus supervisores e colegas de trabalho do Serviço de Padronização e Arquitetura de Software do Tribunal de Contras da União (SEPAS), sem os quais este trabalho não seria possível.

Resumo

O crescimento dos sistemas de informação governamentais gera uma grande demanda por profissionais de TI, mas o processo de contratação de novos desenvolvedores é lento e custoso. Limitações orçamentárias levam cada vez mais órgãos públicos a contratar fábricas de software terceirizadas para suprir a crescente demanda. Mas a introdução de um novo grupo de desenvolvedores ao ambiente e às práticas de desenvolvimento de uma organização pode ser desgastante para os envolvidos no processo de desenvolvimento e entrega dos artefatos de software. Este trabalho descreve o modelo utilizado no Tribunal de Contas da União para realizar o controle do desenvolvimento e aceite de código baseado em técnicas de inspeção contínua. E um sistema criado para facilitar o processo de aceite dos artefatos de software.

Palavras-chave: Qualidade, Métricas, Inspeção Contínua

Abstract

The growth of the information systems within the government causes an increase in the demand for IT professionals, but the process of hiring new developers is slow and expensive. Budget limitations drive an increasing number of organizations to hire third party developers and software factories to supply the soaring demand. But introducing a group of developers to the environment and practices of an organization can be taxing for both parties. This article describes the model for controlling the development and delivery of software artifacts by a third party based on continuous inspection used in the Tribunal de Contas da União (General Accounting Office), and a system designed to ease the process of verifying and accepting the new artifacts.

Keywords: Quality, Metrics, Continuous Inspection

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A falta de profissionais qualificados de TI no Brasil é um problema grave. O apagão de mão de obra qualificada é tão sério que levou o Ministério do Desenvolvimento, Indústria e Comércio Exterior a lançar um censo para medir a extensão da questão e criar políticas para que a falta de profissionais qualificados não afete o crescimento do país a médio e longo prazo [26].

No contexto da administração pública, o cenário é ainda mais precário. O processo de contratação lento e oneroso, dependente da abertura de concurso público, dificulta a expansão dos setores de TI, e a falta de investimento no setor agrava o problema [8]. Uma das soluções mais frequentes para a expansão do parque tecnológico dos órgãos públicos tem sido a terceirização do desenvolvimento de sistemas através de fábricas de software.

Um órgão fundamental da administração pública federal, o Tribunal de Contas da União, é fortemente afetado pela falta de profissionais de TI, o que levou a organização a contratar uma fábrica de software em 2013. O contrato fechado através de licitação previa a entrega de seis mil pontos de função, e a empresa Cast, vencedora da licitação, arrematou no mesmo ano outros três pregões para contratos com o governo, totalizando 15 milhões de reais em contratos [9].

O principal desafio neste cenário é gerenciar os contratos entre o governo e as fábricas de software. Para isso, existem diversas recomendações e documentos, que dispõem regras sobre o processo de contratação e acompanhamento do desenvolvimento de soluções de TI. Entre eles, a instrução normativa número 4 da Secretaria de Logística e Tecnologia da Informação do Ministério do Planejamento, Orçamento e Gestão, a qual dispõe sobre o processo de contratação de soluções de tecnologia da informação pelos órgãos da administração pública [33]. E, mais recentemente, o guia de aquisição do projeto MPS.BR, Melhoria do Processo de Software Brasileiro, o qual descreve um processo de aquisição de software baseado na Norma Internacional ISO/IEC 12207:2008 [32]. Essas recomendações, bem como a experiência do TCU na contratação de serviços de desenvolvimento terceirizado, serão exploradas no Capítulo 3.

Um aspecto frequentemente negligenciado em contratos para desenvolvimento terceirizado é a qualidade interna do produto de software entregue, que muitas vezes é sacrificada em nome da diminuição dos custos e cumprimento de prazos, mas que para o órgão contratante é vital. A qualidade interna afeta diretamente o processo de evolução e manutenção de software, fase que gera mais custos no ciclo de vida de um software[19], o que é especialmente válido para sistemas governamentais, os quais, dada a larga base de usuários

e a natureza conservadora do ciclo de desenvolvimento de software nos órgãos públicos, tendem a ter um ciclo de vida mais longo.

Para auxiliar a avaliação da qualidade interna, existem diversas métricas bem estabelecidas na literatura, cuja medição foi demonstrada eficaz em verificar desvios de qualidade. As principais métricas de *design* para linguagens orientadas a objetos são apresentadas no capítulo 2.

A aplicação dessas métricas, quando prevista em contrato, provê ao gerente de projeto da organização contratante uma visão profunda sobre a estrutura de um projeto sem a necessidade de realizar uma leitura detalhada do código. Mas a inclusão da avaliação de métricas ao processo de desenvolvimento não é trivial, sendo ideal que as medidas sejam adaptadas às necessidades da organização, que seus limiares sejam bem definidos, customizados de acordo com as características do projeto avaliado, e que a coleta e avaliação das métricas seja simples, de forma a ser feita iterativamente durante o desenvolvimento. A qualidade interna, suas métricas e processos de controle são objeto de estudo do capítulo 2.

Existem diversas ferramentas para apurar e avaliar essas métricas no momento da homologação dos artefatos entregues pela fábrica. Entre essas ferramentas, destaca-se a plataforma de inspeção contínua SonarQube (seção 3.4), que vem ganhando popularidade principalmente entre projetos *open source*.

A plataforma, que já vinha sendo utilizada pela equipe de testes e qualidade interna do TCU para acompanhar o desenvolvimento dos sistemas internos, foi adotada para facilitar o processo de homologação e aceite através da análise das métricas. Todavia, à medida que o contrato avançou, o número e a complexidade das ordens de serviço cresceram, de forma que foi necessário otimizar os processos envolvidos no contrato.

1.1 O problema

No contrato entre o TCU e a fábrica de software Cast, aceite era o processo mais trabalhoso do ciclo de desenvolvimento, pois requeria que os artefatos entregues fossem avaliados cuidadosamente, mas, dado o tamanho das entregas, seria inviável avaliar o código linha a linha, e as restrições de tempo e pessoal que levaram o órgão a contratar a fábrica em primeiro lugar impossibilitavam que servidores fossem alocados exclusivamente para lidar com o processo de aceite. Portanto, a análise detalhada da implementação ficaria reservada a componentes críticos, enquanto outros artefatos seriam avaliados somente através de suas métricas e testes. Mas coletar essas métricas era em si só um processo oneroso, pois as medidas deveriam ser consideradas em diferentes níveis dentro do projeto e, ao longo do desenvolvimento do contrato, a aplicação das métricas foi modificada de acordo com negociações entre o órgão e a fábrica.

No início do processo de aceite, as classes alteradas e criadas durante a ordem de serviço eram listadas ao gerente de projeto, que, por sua vez, buscava na plataforma de inspeção contínua SonarQube os arquivos correspondentes e realizava as comparações com o código original ou com os limites estabelecidos para novas classes. As imagens a seguir ilustram os passos necessários para avaliar as métricas de uma classe dentro de um projeto.

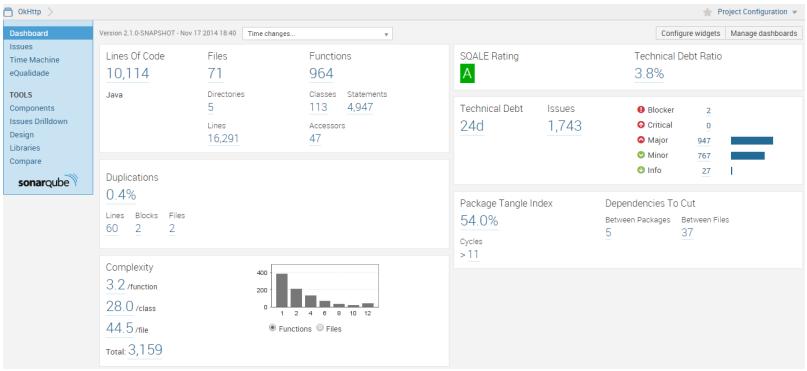

Figura 1.1: Dashboard padrão do SonarQube mostrando as métricas de um projeto *open source* baseado em java.

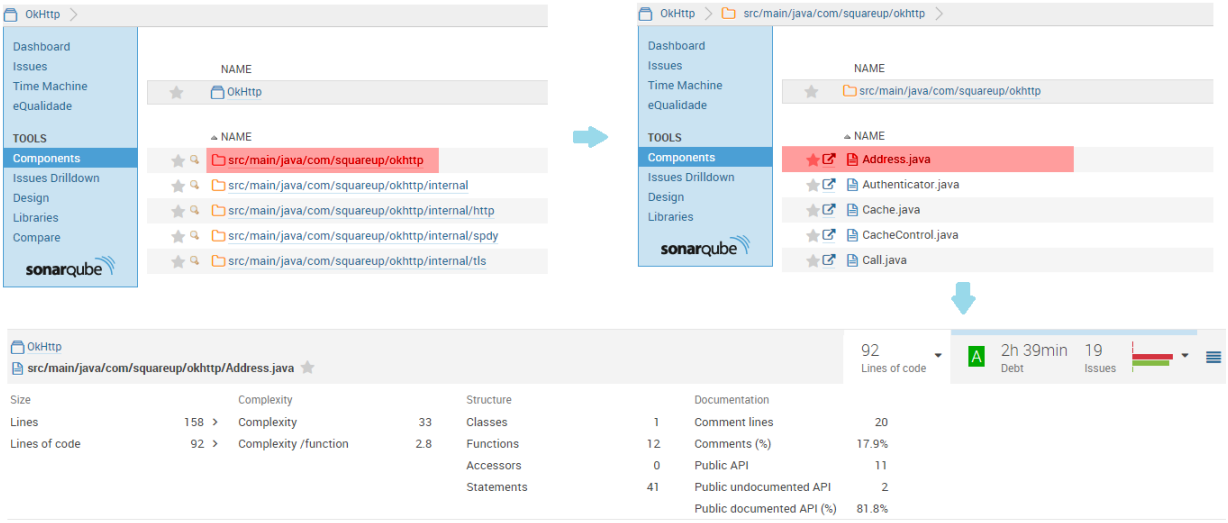

Figura 1.2: Sequência de comandos para acessar as métricas de uma classe na plataforma SonarQube

Como ilustrado na Figura acima, a partir do *dashboard* é preciso acessar o painel de componentes. Neste, navega-se ao componente desejado dentro da estrutura do projeto. As métricas estão disponíveis juntamente ao código da classe. O processo deve ser repetido no projeto *branch* caso seja necessária uma comparação.

Dada a dificuldade de avaliar um grande número de componentes, era essencial a implementação de uma ferramenta para gerar a visualização necessária ao processo de aceite de grandes demandas. O sistema criado para lidar com esse problema será descrito em detalhes no Capítulo 4.

1.2 Objetivos

Dada a importância da avaliação de qualidade dos sistemas governamentais, o objetivo deste trabalho é estender a funcionalidade da plataforma SonarQube de forma a auxiliar a tomada de decisão dos gerentes de projetos no momento da homologação de artefatos de software produzidos por desenvolvedores terceirizados. Para tanto, os seguintes objetivos preliminares foram traçados:

- Revisar a literatura sobre qualidade de software e métricas orientadas a objetos.
- Analisar o processo de controle de qualidade de código do Tribunal de Contas da União.
- Escutar as sugestões dos envolvidos no contrato, incluindo desenvolvedores da fábrica de software e funcionários do TCU.

Capítulo 2

Qualidade de Software

Existem muitas correntes de pensamento sobre o que é qualidade no contexto do desenvolvimento de software. São populares as definições orientadas aos requisitos, tais como: "Qualidade é a capacidade de um produto de software de estar em conformidade com os seus requisitos [13]". Outras definições levam em conta o valor gerado para o cliente [12]. Stephen H. Kan propõe a divisão do conceito de qualidade entre duas visões contrastantes: a visão popular, na qual a qualidade é um atributo imensurável e subjetivo, e a visão profissional, que preza pela conformidade aos requisitos, implicando que estes devem ser bem definidos e sem ambiguidades, e pela adequação ao uso, conceito que leva em conta as expectativas do cliente [16].

Uma das definições mais abrangentes [18] divide a qualidade em cinco perspectivas complementares: Perspectiva Transcendental, que define a qualidade como o ideal que buscamos atingir, um alvo intangível, o qual o processo de aferição de qualidade busca alcançar. Perspectiva do Usuário, que varia de acordo com a percepção pelo cliente da utilidade do produto. Esta visão é moldada pela experiência do cliente com o produto em uso, e, portanto, é largamente subjetiva. Perspectiva de produção, focada no processo de fabricação do produto. Visa diminuir custos de refatoração durante a produção. Analisa a produção de forma independente do produto, focando na adequação dos processos de produção a padrões bem estabelecidos, como o ISO 9001. Perspectiva de Produto, independente da adequação a requisitos, avalia o produto por suas características internas. Provê uma visão objetiva e disponível a partir do início do desenvolvimento e pode suportar a visão de produção por meio de validações sucessivas da estrutura do produto. Não garante que o artefato seja adequado ao uso, mas existem modelos teóricos na literatura ligando a qualidade estrutural interna à qualidade externa (adequação aos requisitos, valor para o cliente). E, finalmente, a perspectiva de valor, que equaciona a qualidade com o valor que o cliente está disposto a pagar pelo produto. Devem ser consideradas as trocas entre a qualidade, o preço e o tempo de entrega do produto, bem como as consequências de uma mudança nos requisitos originais e no orçamento do projeto.

A perspectiva escolhida para definir qualidade também define o método utilizado para medi-la, bem como o momento mais oportuno para realizar a análise. Perspectivas orientadas ao valor utilizam métricas que variam de acordo com o mercado e as respostas do cliente, e são portanto aplicáveis apenas no momento de uma entrega, seja parcial ou final. Visões orientadas ao produto se importam apenas com a estrutura interna e aspectos técnicos da produção, sendo aplicáveis desde o início do processo de fabricação. Perspectivas

de produção lidam com medidas de enconformização aos modelos de processo, contagens de defeitos e custos de retrabalho e, portanto, são melhor aplicadas a estágios avançados de desenvolvimento.

Em comum entre as definições existe a necessidade de quantificar um valor aparentemente subjetivo para operacionalizar a sua análise e a presença do cliente como parte da definição. O cliente é o principal *stakeholder* no processo de avaliação de qualidade de software. Os critérios por ele definidos determinam se um software será aceito e visto como um artefato de qualidade ou se será rejeitado. Cada cliente define uma série de parâmetros pelos quais o produto será avaliado. A transmissão desses parâmetros ao desenvolvedor no processo de levantamento de requisitos incorre na possibilidade de erro. É fundamental que o cliente defina o mais precisamente possível os requisitos para o projeto, de forma a evitar ambiguidades e facilitar o processo de avaliação e aceite.

No contexto da produção de software através de desenvolvedores terceirizados, a perspectiva do cliente toma uma dimensão mais técnica do que o usual, já que o primeiro cliente é o setor de TI da empresa contratante. São incorporadas ao valor do produto a qualidade interna e parte dos valores que compõem a perspectiva de produção, além dos outros fatores supracitados.

2.1 Características Desejáveis em Um Produto de Software

Desde os primeiros estudos na área de medições quantitativas de qualidade de software, o foco dos estudos de qualidade interna é garantir a presença de certas características o mais cedo possível no ciclo de desenvolvimento. Segundo Boehm [2] o valor de um artefato de software, e, portanto, sua qualidade, depende de três grupos de características:

1 **Portabilidade:** habilidade de utilizar o artefato de software independentemente de hardware ou plataforma.

2 **Utilidade:** composta por diversos outros fatores como confiabilidade e eficiência.

3 **Manutenibilidade:** habilidade de estender, modificar e inspecionar um artefato de software.

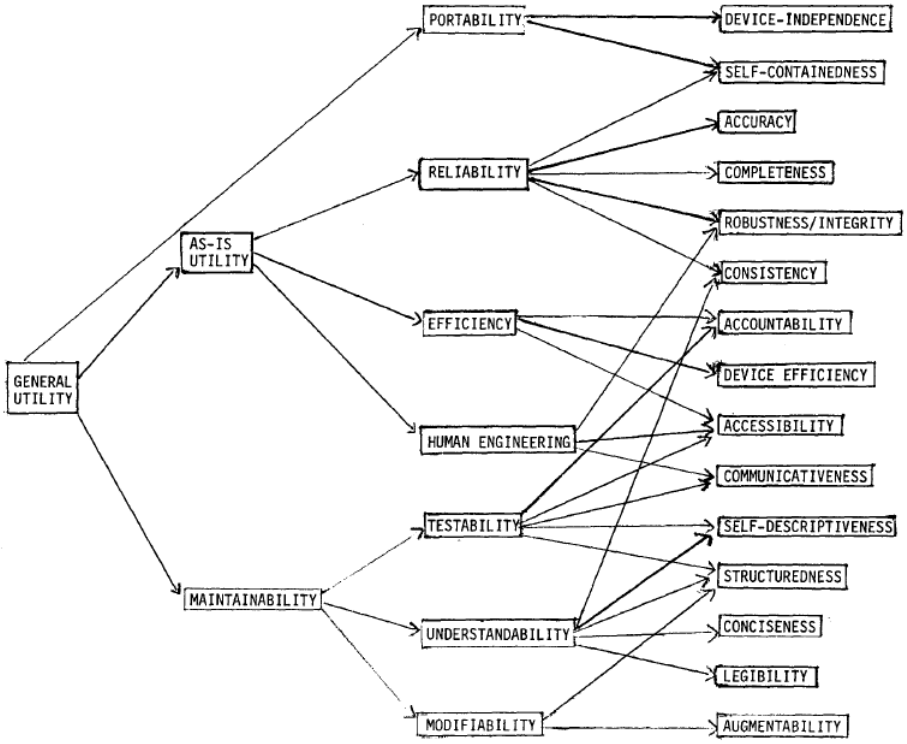

Figura 2.1: Árvore das características da qualidade de software segundo Boehm [2]

Outros autores consideram outros conjuntos de características como sendo componentes principais da qualidade estrutural de um artefato de software. Entre elas, são recorrentes as seguintes características:

2.1.1 Usabilidade

Definido pela ISO 9241 como a medida na qual um produto pode ser usado por usuários específicos para alcançar objetivos específicos com eficiência, eficácia e satisfação em um contexto especifico de uso [27]. Essa definição está, portanto, condicionada à definição de três conceitos assim definidos:

Eficiência: recursos gastos em relação à acurácia e abrangência com as quais usuários atingem objetivos.

Eficácia: acurácia e completude com as quais usuários alcançam objetivos específicos

Satisfação: ausência do desconforto e presença de atitudes positivas para com o uso de um produto.

A usabilidade ocupa, assim, um papel fundamental na percepção da qualidade de um produto de software pelo cliente, e, como tal, deve fazer parte do processo de avaliação e aprimoramento de qualidade.

Diversos fatores podem ser medidos para avaliar a usabilidade como facilidade de instalação e treinamento, mas a avaliação não pode ser automatizada sem grande custo, pois

requer uma observação sistemática do uso da aplicação em um ambiente controlado. As técnicas de medição mais comuns incluem rastreamento do cursor e da visão do usuários na interface da aplicação [14], treinamento e avaliação com voluntários [28] (tradicionalmente cinco por iteração) e testes A/B , nos quais duas versões similares são apresentadas com uma diferença a qual deseja-se avaliar.

2.1.2 Confiabilidade

A previsão da confiabilidade e qualidade de um artefato de software é uma tarefa vital para o desenvolvimento de qualquer aplicação crítica ao seu domínio e deve ser feita o mais cedo possível no ciclo de desenvolvimento para manter o custo de refatoração baixo. Diversas abordagens para a avaliação de qualidade e confiabilidade já foram propostas na literatura [25][7][35]. As métricas são cruzadas com o histórico de falhas e bugs nos projetos com o objetivo de estabelecer uma relação entre medidas de confiabilidade e a qualidade exterior. Os resultados são positivos, a coleção de métricas estáticas pode apontar prováveis pontos de falha durante o desenvolvimento, e, portanto, deve ser feita em qualquer projeto em que se deseje aumentar a confiabilidade do produto final.

2.1.3 Manutenibilidade

O termo possui diversas definições:

- Facilidade do sistema de se recuperar de falhas
- Probabilidade de se detectar, identificar e corrigir falhas em um dado tempo
- Facilidade com que um sistema de software ou componente pode ser modificado para corrigir falhas, melhorar o desempenho e outros atributos ou se adaptar a mudanças no ambiente. [6]

A manutenibilidade é um dos fatores da qualidade interna mais facilmente medidos. Existem na literatura diversos exemplos de tentativas de avaliar por métricas estáticas a manutenibilidade de artefatos de software [21][6][30][36]. A facilidade de coletar e analisar essas métricas de forma automatizada é o principal fator que motiva a implementação de ferramentas de controle de qualidade focadas em manutenibilidade.

2.1.4 Qualidade Interna em Sistemas Governamentais

Para uma órgão do governo, a qualidade dos seus sistemas é uma preocupação constante. A disponibilidade, desempenho e usabilidade dos sistemas são a medida pela qual a população avalia os serviços de software dos órgãos públicos. No entanto, estes fatores são o reflexo de um trabalho intenso de qualidade interna.

Sistemas governamentais geralmente prestam serviços a grandes quantidades de usuários por um longo período de tempo. Assim, a manutenibilidade e confiabilidade são características importantes a esses projetos. Um eventual *downtime* pode custar caro dependendo da utilidade do sistema, logo é necessário avaliar a confiabilidade do sistema para prever esses riscos. Essas características podem ser avaliadas desde o desenvolvimento através da aplicação de métricas de design voltadas para linguagens orientadas a objetos.

2.2 Métricas de Qualidade de Software em Linguagens Orientadas a Objetos

O termo métrica é definido por [2] como a medida da extensão ou grau em que um produto possui ou exibe uma certa característica. Métricas surgem da necessidade de avaliar um artefato objetivamente e são utilizadas no desenvolvimento de software para assegurar fatores como a manutenibilidade. Esta seção apresenta diversas métricas propostas na literatura a fim de escolher um conjunto de métricas que seja compacto, de fácil entendimento e de alto valor para a avaliação de qualidade de um artefato de software.

2.2.1 Suíte de Chidamber-Kemerer

Um dos principais conjuntos de métricas da literatura é a Suíte de Chidamber-Kemerer. Proposta em 1994 para avaliar aspectos da qualidade interna de artefatos construídos com linguagens orientadas a objetos [5], esta suíte de métricas se propõe a medir aspectos relativos à estrutura com rigor matemático, em contraste com as métricas existentes na época, voltadas para a programação estruturada, sem fundamentação matemática e que mediam aspectos relacionados à funcionalidade e não à estrutura. Os autores traçaram os objetivos no artigo inaugural da suíte de métricas:

- Propor métricas que são construídas com uma base sólida nos princípios teóricos de medição na ontologia dos objetos, que incorporem a experiência profissional dos desenvolvedores de software
- Avaliar as métricas propostas contra os critérios de validade estabelecidos
- Apresentar dados empíricos de projetos comerciais para ilustrar as características das métricas em aplicações reais, e sugerir como as métricas podem ser utilizadas

O resultado da pesquisa foi o seguinte conjunto de métricas:

WMC **Weighted Methods per Class:** Em uma classe com n métodos, sendo

$$c_i, ..., c_n$$

a complexidade dos n métodos o valor da métrica é dado da seguinte forma:

$$WMC = \sum_{i=1}^{n} c_i$$

A medida da complexidade por classe serve como indicador para o nível geral de modularização do programa. Um valor alto indica que existem classes muito complexas ou poucas classes com complexidade considerável. A divisão das classes, quando o modelo permite, é recomendável neste caso para aumentar a manutenibilidade.

DIT **Depth of Inheritance:** Tamanho do maior caminho entre a raiz da árvore de herança e a classe analisada. Raramente atinge valores muito altos, devido à natureza da maioria dos sistemas modelados em OO. Utilizada para avaliar o impacto geral

da classe no sistema, aliada à medida do número de herdeiros, a métrica indica a quantas classes se estenderá uma mudança feita em um nível.

NOC **Number of Children:** Quantidade de classes que herdam ou implementam a classe analisada. É utilizada em conjunto com outras métricas para medir a influência da classe dentro do sistema. Idealmente deve ser utilizada para marcar classes de grande impacto no sistema, nas quais os limites de outras métricas devem ser mais estritos.

CBO **Coupling Between Objects:** Medida a partir da equação:

$$CBO = \frac{NumeroDeDependencias}{NumeroDeClassesNoPacote}$$

Uma dependência é caracterizada pelo uso de um método ou variável de outra classe do mesmo pacote, mas apenas uma dependência é contada por classe relacionada.

Um valor alto de acoplamento é indesejável e nocivo à modularidade do projeto, fazendo com que alterações e correções levem mais tempo para ser implementadas e estejam mais propensas a erro, já que as interações entre a classe modificada e suas classes acopladas têm de ser levadas em conta no desenvolvimento e nos testes. Por esse motivo, a métrica incorre em débito técnico (seção 2.2.3) no projeto.

Vale notar que a métrica pode ser calculada em diversos níveis de organização, incluindo pacote, projeto, funcionalidade, revisão no SCM, ou sprint de desenvolvimento.

RFC **Response for a class:** A métrica RFC, Response For Class, é definida como o número de métodos e construtores distintos invocados por uma classe. Pode ser utilizada para medir o impacto geral da classe no sistema, provendo um panorama para a mudança causada pela classe no contexto em que está inserida.

A medida é frequentemente considerada na aplicação da cobertura de testes e é calculada por classe para detectar a necessidade de dividir classes com impacto muito grande, a fim de facilitar a manutenção do módulo.

Uma das limitações da métrica é a medida independente do papel da classe no sistema. Classes que modelam objetos do tipo DTO(Data Transfer Object), por exemplo, fazem muitas chamadas distintas mas possuem pouco impacto no sistema após serem validadas.

LCOM **Lack of Cohesion in Methods:** A métrica possui diversas versões, mas seu cálculo, como proposto em 94 por Chidamber e Kemerer, é feito da seguinte forma:

Seja C uma classe com n métodos, seja In o conjunto de variáveis de instância utilizas pelo método n, seja P o conjunto tal que:

$$P = \{(I_i, I_j) \mid (I_i \cap I_j) = \emptyset\}$$

e o conjunto Q tal que:

$$Q = \{(I_i, I_j) \mid (I_i \cap I_j) \neq \emptyset\}$$

O LCOM é definido como:

$$LCOM = |P| - |Q| \, se \, |P| > |Q|$$

ou zero caso contrário.

Um valor alto na métrica indica falta de coesão dentro da classe, pois seus métodos tratam conjuntos diferentes de variáveis de instância, indicando uma classe com múltiplas responsabilidades

2.2.2 Suíte MOOD

Outro grupo de métricas bem estabelecido é a Suíte MOOD. Foi criada com o intuito de medir a qualidade em sistemas orientados a objeto através de uma avaliação quantitativa [1], de forma a estimar a qualidade exterior do artefato de software analisado. A suíte MOOD é composta por oito métricas definidas segundo sete critérios, São eles:

1 **Métricas devem ser definidas formalmente:** o objetivo é eliminar a subjetividade. Medições diferentes sobre o mesmo artefato devem retornar os mesmos resultados.

2 **Métricas que não avaliem o tamanho devem ser independentes de tamanho:** o valor de uma métrica é ampliado pela habilidade de medi-la em um grande número de projetos, de forma a estabelecer comparações. Naturalmente, o tamanho dos projetos é variado e, portanto, o valor das métricas não deve ser distorcido para que comparações possam ser feitas.

3 **Métricas devem ser adimensionais ou expressas numa unidade de medida consistente:** novamente o objetivo é eliminar a subjetividade nas medidas, rejeitando resultados discretizados como "alto"e "baixo".

4 **Métricas devem ser analisáveis o mais cedo possível no ciclo de desenvolvimento:** a motivação é a redução do custo de refatoração caso seja identificada uma deficiência.

5 **Métricas devem ser escaláveis**: para facilitar a análise do código por diferentes times de desenvolvimento, as métricas devem ser aplicáveis a diferentes níveis dentro do projeto.

6 **Métricas devem ser facilmente computáveis:** o processo de avaliação de qualidade deve ser feito de forma iterativa e, para tanto, é preciso que as métricas sejam fáceis de obter, e que a coleta seja feita preferencialmente de forma automatizada.

7 **Métricas devem ser independentes da linguagem:** para analisar a qualidade da modelagem orientada a objetos as métricas não devem se apoiar em conceitos específicos à uma linguagem, e sim nos fundamentos de orientação a objetos.

Para atender aos critérios estabelecidos, foram propostas 8 métricas com características similares: todas são baseadas em conceitos da orientação a objetos, definidas formalmente através de cálculos simples, assim atendendo aos itens 1, 6 e 7. Todas as métricas são

relativas ao escopo na qual são aplicadas, e definidas como uma razão entre os valores observados no código e o máximo valor possível para a característica em questão e, portanto, atendem aos critérios 2, 3, 4, 5.

As métricas são divididas de acordo com as características da orientação a objetos às quais são destinadas a medir: Encapsulamento, Herança, Polimorfismo e Acoplamento.

MHF **Method Hiding Factor:**

seja:

$$M_v\left(C\right)$$

O número de métodos visíveis em uma classe C. seja:

$$M_e\left(C\right)$$

O número de métodos encapsulados em uma dada classe C.

O total de métodos na classe é calculado como:

$$M_t\left(C\right) = M_v\left(C\right) + M_e\left(C\right)$$

O fator de encapsulamento de métodos representa o uso de encapsulamento no código em relação ao potencial para encapsulamento no programa e é representado por:

$$MHF = \frac{\sum_{TC}^{i=1} M_h\left(C_i\right)}{\sum_{TC}^{i=1} M_t\left(C_i\right)}$$

onde TC é o total de classes analisadas.

AHF **Atribute Hiding Factor:**

Alternativamente é possível definir uma medida similar para os atributos da classe analisada.

$$AHF = \frac{\sum_{TC}^{i=1} A_h\left(C_i\right)}{\sum_{TC}^{i=1} A_t\left(C_i\right)}$$

MIF **Method Inheritance Factor:**

Seja:

$$TM_h$$

o total de métodos herdados, calculados da base da árvore de herança até a classe em questão. E seja

$$TM_d$$

o total de métodos disponíveis na classe, representado pelo número de métodos herdados somado ao número de métodos novos definidos. O fator de herança de métodos é dado por:

$$MIF = \frac{TM_h}{TM_d}$$

AIF **Atribute Inheritance Factor:** Analogamente é definida a mesma métrica para atributos pela relação:

$$AIF = \frac{TA_h}{TA_d}$$

COF **Coupling Factor:** Para definir o acoplamento, é preciso definir a relação de cliente e servidor entre classes. Uma classe

$$C_c$$

é cliente de uma classe

$$C_s$$

sempre que contiver uma referência a um método ou atributo da classe servidora. O coeficiente de acoplamento é definido da seguinte forma:

$$COF = \frac{\sum_{TC}^{i=1} \left[\sum_{TC}^{j=1} ehCliente(C_i, C_j) \right]}{TC^2 - TC}$$

PF **Polimorfism Factor:** A dificuldade no cálculo de um coeficiente de polimorfismo é identificar quantas relações polimórficas são possíveis no código analisado. O mínimo de polimorfismo é atingido quando nenhum método é sobrecarregado e o máximo é atingido quando todos os métodos de todas as classes não raiz de sua própria árvore de herança são sobrescritos. Se um método M for sobrescrito em todas os descendentes de uma classe C, uma chamada a M pode ter NOC(C) destinos possíveis. Analogamente o cálculo pode ser feito para todo o sistema:

$$\sum_{TC}^{i=1} \left[M_d\left(C_i\right) * DC\left(C_i\right) \right]$$

o cálculo do coeficiente de polimorfismo leva em conta o total de relações polimórficas possíveis contra o valor das relações existentes e é calculado da seguinte forma:

$$PF = \frac{\sum_{i=i}^{TC} \left[\sum_{j=1}^{DC(C_i)} M_o\left(C_j\right) \right]}{\sum_{i=1}^{TC} \left[M_d\left(C_i\right) * DC\left(C_i\right) \right]}$$

2.2.3 Métricas Analisadas na Plataforma SonarQube

A plataforma de inspeção contínua SonarQube (seção 3.4), utilizada no TCU, é capaz de realizar o cálculo de diversas métricas, incluindo as métricas das suítes supracitadas. Além dessas, outras métricas merecem destaque por proverem informações importantes sobre os artefatos analisados.

- **Complexidade:** contagem dos caminhos possíveis que o fluxo de controle que um método pode tomar. Pode ser calculado a partir da representação do programa como um grafo [23], representando instruções como nós e o fluxo de controle como arestas, um bloco de decisão como um comando *if* ou um bloco *while* divide o fluxo de controle do método em dois. A partir do grafo, é possível calcular a complexidade da seguinte forma:

  $$NumeroDeArestas - NumeroDeNos + 2$$

  A complexidade é um dos principais fatores na avaliação da manutenibilidade de qualquer artefato de software e tende a ser concentrada fortemente nas camadas de negócio das aplicações. A presença de valores altos de complexidade na camada de persistência e apresentação são indicativos de uma arquitetura mal planejada. Limitações da tecnologia utilizada também podem afetar a distribuição da complexidade no projeto.

  O aumento da complexidade requer um esforço significativo da equipe de testes para que estados possíveis no programa não fiquem sem cobertura. A complexidade possui uma ligação direta com a cobertura de testes, criando a necessidade de considerar a cobertura dos possíveis caminhos de controle no programa separada da cobertura geral, garantindo uma melhor distribuição dos testes. Essa necessidade é suprida pela a adição da medida de cobertura por caminho (*branch coverage*).

- **Índice de Emaranhamento de Pacote**: Métrica composta que mede de 0 a 100 o esforço necessário para remover ciclos de dependências entre pacote [22]. Aplica-se especialmente bem a arquiteturas em camadas com regras restritivas de acesso. Um valor baixo indica que não existem dependências cíclicas entre os pacotes do projeto, ou que a dependência é pontual e o esforço necessário para eliminá-la é baixo. Um valor alto indica um grande esforço de refatoração necessário para retirar os ciclos de dependência. A métrica afeta diretamente o débito técnico do projeto.

- **Linhas Duplicadas:** Importante para evitar a propagação de erros por cópia de código antigo. Um valor alto é indicador de uma solução mal planejada, o que dificulta o reuso de código e a manutenção. Outra causa possível é uma limitação da tecnologia utilizada. A métrica contribui para o débito técnico e aumenta as chances de violações à arquitetura se propagarem do código legado para novos projetos.

- **Cobertura de Testes de Unidade e de Integração:** importante para a manutenibilidade e confiabilidade do sistema. Além de assegurar o bom funcionamento da aplicação, a cobertura de testes é importante para verificar os requisitos e facilitar a manutenção e evolução do projeto [37]. A métrica sofre uma limitação primordial na medição da distribuição da cobertura. Se a cobertura de testes for concentrada em um conjunto de artefatos que não realizam operações de negócio como os DAOs (Data Access Object), o valor da métrica não se traduz em qualidade no projeto final.

  É preciso estabelecer regras para a cobertura baseadas na arquitetura do sistema, focando o esforço de testes na camada de negócios e nos artefatos mais importantes.

Um controle rígido da nomenclatura dos artefatos pode facilitar aplicação desta métrica.

- **LCOM4:** A métrica LCOM4 é uma variação da métrica LCOM descrita na suíte de Chidamber-Kemerer. É utilizada para checar se a classe analisada obedece o princípio da única responsabilidade [29]. A métrica é calculada a partir do uso das variáveis locais por bloco de código. Um bloco é definido a partir dos delimitadores da linguagem e as chaves são utilizadas para delimitar os blocos no caso do java. O valor da métrica é incrementado em um para cada grupo isolado de blocos utilizando um conjunto de variáveis locai. As figuras a seguir ilustram o calculo do LCOM4:

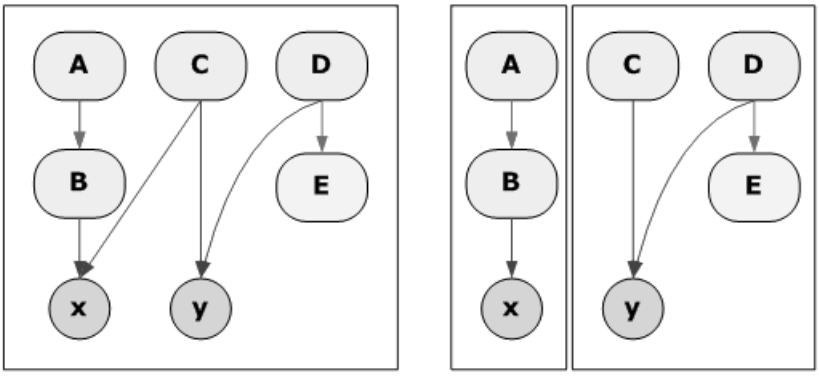

Figura 2.2: Representação em grafo do uso de variáveis entre compartilhadas entre métodos.

Na figura à direita, os métodos A e B utilizam a variável local X, constituindo um grupo, e os métodos C, D e E utilizam apenas a variável Y. A métrica considera que as variáveis estão sendo utilizadas com propósitos diferentes, o que viola o princípio da única responsabilidade dentro da classe. O valor da métrica, neste caso, é dois. Na imagem à esquerda, os blocos utilizam as mesmas variáveis locais x e y, estando assim no mesmo grupo. Em uma classe em que todos os blocos são usados para modificar o mesmo conjunto de variáveis locais, o princípio da única responsabilidade tem mais chances de estar sendo respeitado. O valor, neste caso, é um.

- **Débito Técnico:** Métrica fundamental para a visualização da saúde do projeto pela gerência, faz uma conversão do esforço necessário para corrigir as falhas arquiteturais do projeto em um custo baseado no preço da homem-hora praticado na organização [3]. A metáfora da dívida inclui o pagamento de juros, na forma de homem-horas gastas no desenvolvimento devido ao peso das falhas arquiteturais, falhas como alto acoplamento, que dificultam a evolução do projeto e fazem com que novas funcionalidades levem mais tempo para ser implementadas.

  As fontes de débito técnico podem ser de natureza intencional ou não-intencional. Fontes intencionais incluem todas as práticas adotadas com o intuito de diminuir os custos ou aumentar a produtividade em detrimento das boas práticas de qualidade de software. Fontes não-intencionais incluem diversos fatores, como decisões de projeto

propensas a erros, adoção de novos componentes com débito técnico acumulado e a descoberta de vulnerabilidades em componentes utilizados, entre outros aspectos fora do controle ou do conhecimento da equipe.

A medida do débito técnico deve ser adaptada para a organização interessada em controlá-lo, tanto no valor da homem-hora como nos fatores que aumentam o débito, como violações pontuais à arquitetura, valores altos em métricas de acoplamento, erros de projeto que dificultam a implementação de novas funcionalidades, entre outros.

2.3 Aplicação das Métricas no Contrato

A listagem destas métricas serviu ao primeiro objetivo preliminar: "Revisar a literatura sobre qualidade de software e métricas orientadas a objetos". Entretanto, a aplicação das métricas ao contrato do TCU com a fábrica passa por um processo de negociação. O conjunto final das métricas aplicadas ao contrato varia de acordo com estas negociações e portanto o processo de aceite deve se adaptar a estas modificações.

2.4 Validação Empírica das Métricas

A literatura propõe diversos modelos para validar empiricamente o mérito do uso das métricas e observar a correlação entre bons valores nas métricas e seus efeitos na manutenibilidade, confiabilidade, usabilidade e eficiência [31][34][11].

A abordagem baseada em métricas possui, portanto, a fundamentação necessária para ser uma fonte confiável de informação sobre a qualidade de um artefato de software, de forma a prover uma visão confiável sobre a qualidade do produto.

A avaliação dessas métricas no contrato entre o Tribunal de Contas da União e a fábrica de software Cast é condicionada ao tipo de ordem de serviço contratada. A fim de entender a dinâmica do contrato e o papel das métricas na homologação de cada ordem de serviço, o capítulo seguinte apresenta as regras e recomendações dispostas pelo Tribunal durante o processo de contratação da fábrica de software.

Capítulo 3

Contratação de TI no Tribunal de Contas da União

A contratação de uma fábrica de software é um processo complexo para qualquer organização. Para o Tribunal de Contas da União, a contratação da fábrica de software Cast se deu após um longo processo que transcorreu entre 30/04/2012 e 08/05/2014. O valor global do contrato, como descrito na proposta vencedora da licitação, foi de R$6.883.752,29. Dado o alto custo para o governo nas contratações de fábricas de software terceirizadas, é preciso adotar critérios rígidos para a avaliação dos resultados dessas empresas. Este capítulo apresenta as principais recomendações e normas descritas no Termo de Referência para Contratação de Serviços de Desenvolvimento e Manutenção de Sistemas, documento integrante ao processo e que descreve as práticas, processos e metas para o bom andamento do contrato, bem como as expectativas da instituição contratante para com o serviço prestado.

A formalização do processo dentro da entidade ocorreu através das normas dispostas pela portaria-CGTI número 1, de 11 de outubro de 2013, que cita como motivação para a formalização do processo de contratação a necessidade de aperfeiçoar o processo de trabalho, propiciando uma redução nos custos e no tempo de execução dos procedimentos. O guia de referência do processo lista como objetivo para a contratação ampliar a capacidade de entrega da área de tecnologia da informação do Tribunal de Contas da União e, ao mesmo tempo, assegurar a qualidade dos produtos entregues para as demais áreas do Tribunal de Contas da União.

3.1 Papeis no Contrato

A complexidade do contrato recai na necessidade de assegurar a qualidade das entregas, e, para tal, são definidos papéis chaves, a serem desempenhados tanto por servidores do órgão quanto por representantes da fábrica, a fim de garantir a comunicação entre o órgão e o contratado, permitindo, assim, o controle da qualidade dos artefatos entregues.

- **Gerente de Contrato:** apontado pela fábrica, é responsável pela interlocução técnica com o TCU acerca da qualidade e do andamento dos serviços. Deve zelar pela qualidade geral dos serviços prestados pela contratada, participar das reuniões regulares de acompanhamento do contrato em periodicidade definida pelo TCU,

avaliar, em conjunto com o TCU, os níveis de serviço alcançados pela contratada e negociar as medidas corretivas em caso de problemas na execução de uma ordem de serviço.

- **Gerente de Demandas:** o gerente de demandas da contratada é responsável pela comunicação com o TCU acerca do andamento das OS não classificadas como projeto. Suas tarefas são as seguintes: realizar e apresentar ao TCU o planejamento de atendimento das OS encaminhadas para a contratada, gerenciar a equipe designada para execução das OS sob sua responsabilidade, assegurando o comprometimento de todos com os objetivos e níveis de serviço previstos, participar, quando convocado, da reunião de acompanhamento do contrato e responsabilizar-se pelo controle interno de qualidade dos produtos entregues pela contratada.

3.2 Ordens de Serviço

Instituídas como a unidade de prestação de serviço dentro do contrato, as ordens de serviço(abreviadas como OS daqui em diante) são classificadas em seis tipos, cada um com seu próprio fluxo de desenvolvimento e avaliação.

- **OS de projeto:** Ordens de serviço do tipo projeto são as mais complexas previstas no contrato, frequentemente apresentando a necessidade de segmentar o processo de entrega em entregas intermediárias.

  A fábrica deve indicar para cada demanda do tipo projeto um profissional para servir como gerente de projeto. É papel do gerente de projeto:

  - Realizar o planejamento do projeto.
  - Realizar o gerenciamento de riscos do projeto, prevendo problemas, propondo soluções de contenção e identificando oportunidades de refatoração.
  - Gerenciar a aplicação do fluxo de trabalho definido para o projeto.
  - Reportar à organização o andamento do projeto.

  A entrega dos artefatos é feita na forma de *releases* de produção e de homologação. Uma *release* de produção contempla um caso de uso ou funcionalidade que possa ser colocada em produção, e cada release de produção pode conter uma ou mais *releases* de homologação. *Releases* de homologação são conjuntos de funcionalidades prontas para o processo de homologação.

- **OS de manutenção evolutiva/adaptativa:** Demandas são classificadas como manutenção evolutiva/adaptativa quando tratam de alteração ou inclusão de nova funcionalidade em aplicação existente ou desenvolvimento de conjunto de casos de uso que não sejam classificados como projeto pelo TCU.

  Tais demandas requerem que o órgão disponibilize documentação atualizada sobre o sistema alvo. Caso este não tenha sido desenvolvido pela fábrica em uma demanda do tipo projeto, a documentação pode ser requerida na forma de uma OS de documentação de sistemas.

- **OS de documentação de Sistemas:**

  A OS de documentação contempla a documentação de um conjunto de casos de uso de uma solução. O fluxo de trabalho é simples e a remuneração se dá em parcela única.

- **OS de Teste de Sistemas:**

  Tem por objetivo realizar a verificação de qualidade em uma aplicação, bem como tratar da documentação dos testes. A OS pode contemplar toda a aplicação ou apenas alguns casos de uso.

- **OS de Preparação de Ambiente de Treinamento:**

  Esta OS tem por objetivo a configuração de um ambiente incluindo aplicação e banco de dados para que o TCU possa ministrar o treinamento de uma aplicação ao seus usuários. Esse tipo de OS tipicamente acompanha a criação de uma OS do tipo projeto, de forma que a fábrica completa o ciclo de desenvolvimento de um novo sistema com a disponibilização de um ambiente para treinamento no novo sistema.

- **OS de Sustentação de Sistemas:**

  A OS de sustentação (ou manutenção) visa manter o funcionamento normal de um sistema de acordo com os seus requisitos.

  A sustentação engloba, também, investigação e tratamento de incidentes relativos à degradação de desempenho da aplicação ou relativos a erros funcionais. Haverá incidente de degradação de performance quando o tempo de resposta de uma dada requisição for 20% superior ao comportamento habitual desta, considerados o mesmo dia da semana e horário da medição.

  Os serviços contemplados na OS de sustentação são:

  - Investigação de incidentes e diagnóstico de causa
  - Restabelecimento do nível do serviço (solução de contorno)
  - Manutenção corretiva (tratamento da causa raiz/solução definitiva do problema)
  - Suporte à operação da aplicação com a preparação de scripts para sanar situações não tratadas pela aplicação, extrair dados, entre outras situações

3.3 Atividades do Contrato

Para cada cada OS, à exceção da OS de sustentação, o Tribunal determina os grupos de atividades a serem realizados. Os grupos de atividade são:

- Levantamento de requisitos
- Análise e projeto
- Construção
- Testes

- Homologação
- Tarefas específicas do TCU
- Gerenciamento de projetos
- Preparação de ambiente para treinamento

Este trabalho tem interesse especial na atividade de homologação, que é um ponto importante de contato entre o Tribunal e a fábrica de software e está presentes em quase todos os tipos de OS. Para supervisionar esta e outras atividades, a fábrica contratada deve indicar um gerente de contrato, encarregado de zelar pela qualidade geral dos serviços prestados, supervisionar a atuação dos gerentes de projeto e do gerente de demandas da contratada indicados como responsáveis por OS, participar das reuniões regulares de acompanhamento do contrato e avaliar, em conjunto com o TCU, os níveis de serviço alcançados pela fábrica. Essa avaliação é assistida por uma série de ferramentas e processos que já haviam sido estabelecidos no ambiente de desenvolvimento do tribunal e foram adaptados ao ciclo de trabalho da fábrica de software.

A fim de entender a cultura da organização posteriormente ao inicio do contrato com a fábrica, é necessário expor as técnicas e práticas utilizadas no tribunal para garantir a qualidade dos artefatos produzidos.

3.4 Ferramentas de Controle de Qualidade Utilizadas no TCU

- **Checkstyle** Ferramenta de análise estática especializada em código java, capaz de percorrer a árvore sintática das classes realizando verificações estruturais. O checkstyle se integra ao processo de desenvolvimento por meio de um *hook* no servidor de versionamento. Ao realizar um *commit*, os arquivos adicionados ou atualizados passam pela verificação da ferramenta, e o *commit* só é liberado caso os arquivos estejam em conformidade com as normas. São verificadas falhas de padronização de codificação e violações arquiteturais [24]. O objetivo do *hook* é impedir a propagação de falhas na base de código através de cópia.
- **PMD** Ferramenta de análise estática utilizada em menor escala para implementar verificações que não incidem sobre arquivos java. É útil para verificar arquivos de configuração e propriedades de forma simplificada através da linguagem de consulta voltada para XML Xpath [10]. A ferramenta auxilia a geração de relatórios de qualidade, mapeando na base de código ocorrências de configurações depreciadas, subótimas ou que violem a arquitetura, bem como arquivos de propriedades em outros formatos que possam expor informações críticas.
- **Jenkins** Servidor de integração contínua, realiza a construção de todos os artefatos de software e auxilia o processo de publicação [17]. Os testes de unidade são realizados a cada nova construção periódica, alertando para mudanças que venham a introduzir erros no código. O servidor também serve como plataforma de automação de certas tarefas através da execução de scritps e da avaliação das métricas dos artefatos.

- **Wiki** Utilizada como plataforma para disseminar conhecimento e documentar em alto nível os projetos da organização [20], a wiki interna contém informações sobre os projetos e processos envolvidos no desenvolvimento e manutenção dos sistemas corporativos do órgão. O caráter colaborativo da plataforma permite que as equipes de desenvolvimento cataloguem suas decisões de projeto de forma independente, mas com o apoio das experiências de outros times, agrupada em um formato simples e familiar à maioria dos desenvolvedores.

- **SonarQube** Plataforma de inspeção contínua, se integra ao servidor de integração contínua para avaliar e coletar métricas sobre os artefatos em construção [4]. As ferramentas supracitadas se integram ao SonarQube através de métricas customizadas, e portanto as falhas presentes na base de código antes do inicio da aplicação destas ferramentas são identificadas através da análise do SonarQube.

  A avaliação das métricas é utilizada para identificar problemas como ciclos de dependências, acúmulo de complexidade em componentes críticos e má distribuição de cobertura de testes, guiando o processo de manutenção e refatoração. A plataforma assumiu um papel de destaque após a contratação da fábrica de software, a sua natureza extensível permitiu que fossem implementados *plugins* para atender às necessidades da organização durante o contrato.

  Todavia, devido a limitações da plataforma, o processo de aceite dos artefatos de software estava consumindo bastante tempo e recursos, e a necessidade de avaliar diversas métricas em cada componente se traduzia em consultas longas e demoradas à plataforma. Seria necessário, portanto, gerar os relatórios das métricas de outra forma.

Capítulo 4

eQualidade

O Equalidade é um sistema desenvolvido em 2013 no Tribunal de Contas da União com o objetivo de facilitar o processo de aceite de artefatos de software produzidos pela fábrica de software Cast, gerando visões e relatórios refinados sobre a qualidade interna dos sistemas. A apresentação dos resultados é feita em diferentes níveis dentro do projeto analisado, de forma a aplicar as métricas mais apropriadas para cada nível. Os requisitos do sistema evoluíram junto ao processo institucional de aceite de software, adicionando funcionalidades para facilitar a tomada de decisão dos gerentes de TI, bem como a autoavaliação pela fábrica durante o desenvolvimento. Este capítulo apresenta o sistema e descreve suas principais funcionalidades.

A primeira versão da ferramenta foi implementada como um sistema web que consumia e tratava dados provenientes do *webservice* da plataforma SonarQube. A ferramenta era capaz de gerar diversas visões sobre os dados e agregava valor aos dados da plataforma Sonar, gerando comparações e analisando as métricas em níveis não explorados pela plataforma.

4.1 Visões do sistema

4.1.1 Nível de Projeto

Visão mais simples gerada pelo sistema, é possível gerar comparações entre projetos através de métricas pré estabelecidas. As métricas pertinentes ao nível de projeto são comparadas entre projetos ou de acordo com limites pré determinados a cada OS.

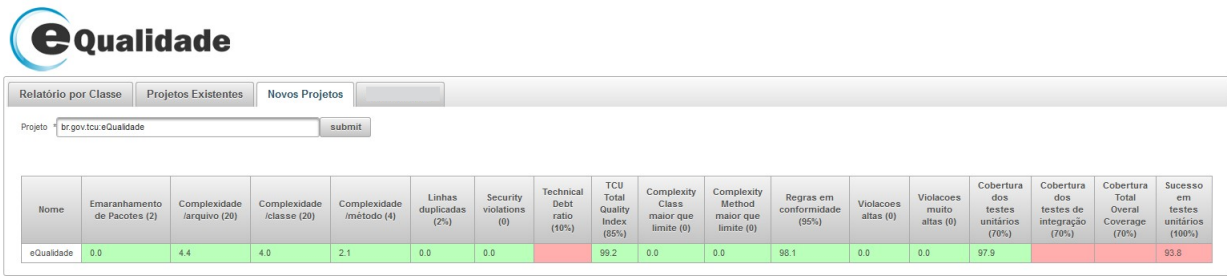

Figura 4.1: Interface do sistema para a avaliação de novos projetos.

O relatório é apresentado visualmente como uma tabela marcando em verde as métricas dentro do limite e em vermelho as violações.

Uma funcionalidade implementada com o intuito de isolar os indicadores de qualidade por funcionalidade foi a visão por pacotes. Devido ao escopo diminuído, a visão em nível de pacote apresenta indicadores mais simples. Métricas como complexidade e cobertura são apresentadas como médias entre os arquivos do pacote, sendo possível visualizar todos os pacotes de um sistema de uma vez. Essa funcionalidade é usada para identificar pontos críticos no sistema relativos a alguma métrica, o que facilita o processo de refatoração. Sua implementação é bastante similar à da visão por pacotes.

4.1.2 Nível de Classe

Visão mais detalhada do sistema, integrada ao sistema de versionamento para facilitar a análise. A visão sobre classes é ideal para o uso pela fábrica durante o desenvolvimento, pois os problemas podem ser identificados iterativamente de forma a avaliar a qualidade das *releases* a cada *commit*.

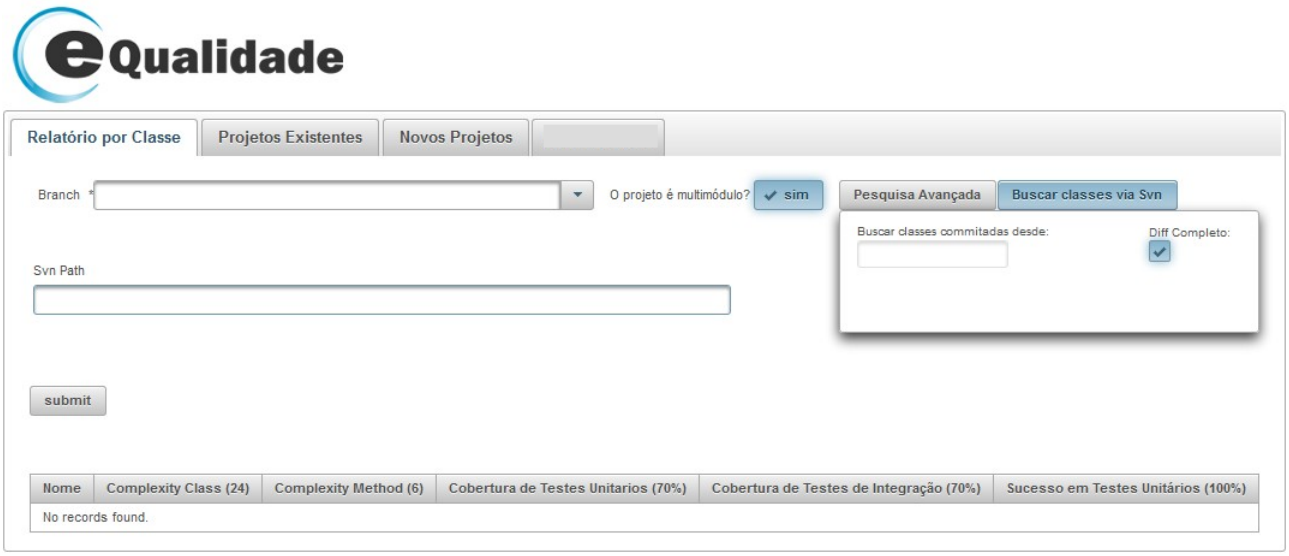

Figura 4.2: Interface do sistema para a avaliação de classes individualmente. As classes modificadas no último *commit* são obtidas via chamada ao cliente do servidor de versionamento.

O eQualidade realiza chamadas ao cliente do servidor de versionamento para que apenas as classes modificadas desde o último *commit* fossem analisadas. Essa funcionalidade requer a presença de um cliente do sistema de versionamento escolhido no servidor que hospeda o sistema, mas provê a capacidade de analisar as classes de forma iterativa. Um desenvolvedor da fábrica ou do próprio órgão pode ter acesso às métricas das classes mais recentemente alteradas, recebendo um *feedback* instantâneo sobre o impacto de seus *commits* nas métricas do projeto. A versão do sistema que está em produção se integra ao Subversion, mas no caso de sistemas de versionamento descentralizados, como o git, é possível analisar as classes do repositório local com versões locais do Sonar e do eQuali-

dade, de forma a verificar a qualidade dos *commits* antes de sua integração ao repositório central.

4.2 Avaliação de Qualidade em Sistemas Legados

A principal diretiva do TCU em caso de sistemas legados é fazer com que a qualidade dos projetos seja ao menos mantida. Adições ao sistema são medidas em relação ao estado original daquele, garantindo a estabilidade das métricas, mas qualquer adição ao código potencialmente introduz complexidade e diminui o percentual de cobertura, fazendo com que em muitos casos a implementação de uma nova funcionalidade deva vir acompanhada de uma refatoração para que sejam mantidos os indicadores. Portanto, demandas mais extensas devem estabelecer limites para as alterações, e a possibilidade de refatoração deve ser levada em conta nas estimativa dos recursos necessários à OS.

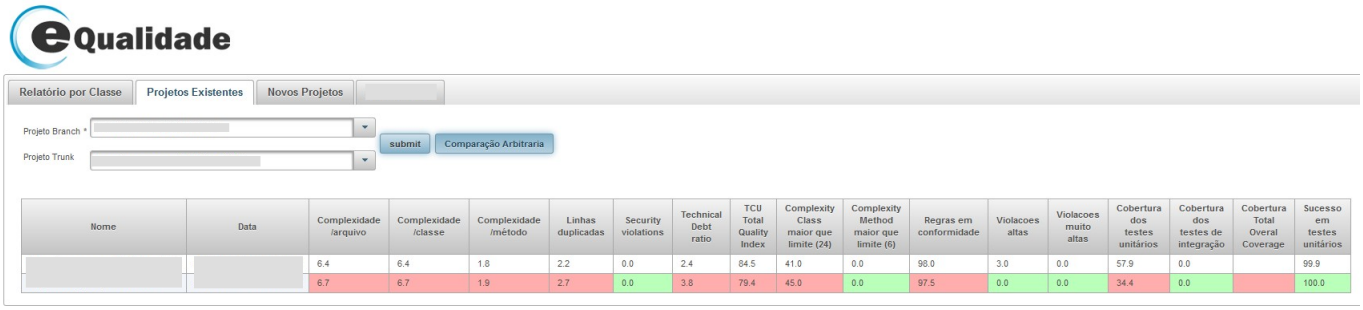

Figura 4.3: Interface do sistema para a avaliação de projetos. É possível comparar uma série de métricas entre dois projetos arbitrários ou deixar que o sistema resolva o nome do sistema principal (trunk) baseado no nome do sistema *branch*

4.3 Utilização e Aceitação

Por ter sido feito sob a demanda dos gerentes, a aceitação foi rápida. O sistema foi disponibilizado localmente e posteriormente migrado para um servidor próprio. A utilização do sistema não ficou restrita aos gerentes. Pouco tempo após o lançamento da ferramenta, esta foi adotada como medida de autoavaliação para os desenvolvedores da fábrica de software, o que motivou a implementação da integração com o sistema de versionamento. A constante comunicação com os gerentes e desenvolvedores causou a evolução do sistema, na forma de ajustes em limites para as métricas e funcionalidades, como a análise de sistemas multimódulo, cuja natureza dificulta a resolução dos nomes dos recursos na plataforma SonarQube.

A principal limitação do sistema é a dependência do uso do webservice público da plataforma SonarQube para localizar os recursos e obter valores das métricas. Embora poderoso, o webservice limita as possibilidades de expansão da ferramenta e dificulta modificações pontuais, já que o formato da resposta varia bastante de um tipo de query para o outro. Outro ponto que deixa a desejar é a logística de instalação do sistema. Pelo fato de o sistema requerer um cliente do SCM na máquina na qual está rodando,

o eQualidade requer que o servidor que o hospeda faça chamadas de alto custo a outros servidores (integração contínua via http para acessar o webservice e versionamento via execução do cliente do SCM), o que levou o sistema a ser utilizado como cópia local pelos gerentes e desenvolvedores da fábrica até que a infraestrutura necessária estivesse liberada e disponível.

Além dos aspectos técnicos, questões conceituais interferiam com a utilização do sistema. Por exemplo, uma das métricas mais controversas quanto a sua análise e aplicação foi a cobertura por testes unitários e de integração. A análise da cobertura a nível de projeto se provou ineficiente, pois a distribuição dos testes entre as classes não era garantida, e classes como DAOs geravam controvérsias sobre a forma de aplicar a métrica. O assunto gerou discussões entre os gerentes da fábrica e do órgão e uma metodologia unificada de aplicação de cobertura não foi definida durante o tempo de vida da primeira versão do eQualidade.

4.4 Requisitos para a nova versão

De acordo com o escopo de aplicação do sistema, as características desejáveis à implementação da nova versão da ferramenta eram as seguintes:

Portável: fácil de integrar ao processo de desenvolvimento.

Extensível: métricas customizadas podem ser adicionadas.

Visual: apresenta as métricas de forma natural ao processo de desenvolvimento para agilizar a identificação de pontos de refatoração.

Flexível: a visualização das métricas pode ser configurada de acordo com as necessidades de cada projeto ou OS.

A portabilidade garante que o sistema possa ser testado em menor escala facilmente, e posteriormente integrado ao processo de desenvolvimento sem a necessidade de modificar o ambiente existente. A implementação como *plugin* do sonar garante essa funcionalidade.

O caráter visual do relatório apresentado torna o processo de identificar falhas mais simples e rápido, e o progresso pode ser verificado iterativamente, tanto pelos gerentes quanto pela fábrica durante o desenvolvimento.

A flexibilidade garante que a organização possa imprimir seus próprios valores no sistema de avaliação de qualidade, priorizando regras e métricas as quais julgarem essenciais.

O potencial de extensibilidade é fundamental para garantir que o sistema acompanhe a evolução do processo de avaliação e desenvolvimento de software na organização, facilitando uma eventual mudança de tecnologia, processo, ou até do modelo de fábrica de software para outro modelo.

Para que o sistema acompanhasse o desenvolvimento do contrato com a fábrica, seria necessário, portanto, uma mudança na estrutura fundamental da ferramenta. A proposta era implementar o sistema como um plugin da plataforma SonarQube, o que daria acesso direto às métricas via querie no banco de dados da plataforma, facilitando bastante a atualização dos relatórios e possibilitando a integração da ferramenta ao processo de trabalho de forma suave, uma vez que a plataforma já estava em uso e o sistema seria integrado como um *widget*, deixando os dados disponíveis a cada análise dos artefatos.

4.5 eQualidade 2.0

Seguindo a proposta, a nova versão do eQualidade foi implementada como uma extensão da plataforma SonarQube. A facilidade de integrar a nova solução ao ambiente pré existente foi um fator chave nessa decisão de projeto, e a capacidade de consultar diretamente o banco de dados da plataforma SonarQube sem a restrição imposta pelo webservice criou diversas oportunidades de customização.

Figura 4.4: Painel de controle do eQualidade no SonarQube, exibindo as possibilidades de customização em nível de projeto. Há um outro painel disponível para configurações globais.

As principais funcionalidades da nova versão implementam diretamente as visões do projeto anterior, incluindo novas possibilidades para a customização da avaliação.

Existem dois modos principais de visualização das métricas na ferramenta: modo de comparação, utilizado para avaliar as demandas evolutivas e de manutenção, e modo de limites, utilizado para novos projetos.

O modo de comparação é ativado via propriedade no painel de configuração. O campo "Projeto Trunk"recebe o nome do projeto alvo da comparação e o campo "Métricas para Comparação"recebe a lista de métricas que serão coletadas dos dois projetos. Vale notar que todas as métricas disponíveis em nível de projeto na plataforma sonar ficam disponíveis para comparação, mas apenas as métricas presentes em ambos os projetos serão apresentadas.

Este modo possui também a opção de realizar a avaliação das classes do projeto, embora seja melhor utilizado através da análise incremental da plataforma. Este modo de análise atualiza as métricas apenas nas classes modificadas desde a ultima avaliação, garantindo que apenas as classes modificadas sejam exibidas no relatório.

O segundo modo, feito para a avaliação dos novos projetos, faz a avaliação das métricas de acordo com os limites customizados no painel de configuração através do campo nomeado "Métricas por Projeto". Este modo conta também com métricas customizáveis. São elas:

- **Métricas condicionadas a outras métricas:** Esta opção permite coletar as métricas em arquivos que passarem de um certo limite em outras medidas. A motivação para a implementação dessa funcionalidade foi possibilitar que classes com certas características, como alta complexidade ou alto número de herdeiros, recebessem atenção especial no processo de aceite.
- **Métricas condicionadas à nomenclatura da classe:** Motivada pela necessidade de aplicar medições em classes com papéis específicos na arquitetura do órgão. Esta funcionalidade permite que classes identificadas pelo nome tenham suas métricas agrupadas no relatório e foi motivada pela necessidade de avaliar classes de acordo com o seu papel no sistema, identificado através de um prefixo.
- **Métricas condicionadas ao conteúdo da classe:** Análoga à funcionalidade anterior, a motivação para esta opção foi a necessidade de avaliar classes contendo anotações específicas da arquitetura do TCU, e que não são necessariamente identificadas via nomenclatura.

Entre os dois modos são contempladas as principais ordens de serviço do contrato e o caráter visual da apresentação dos dados na interface principal do plugin.

4.6 Utilização e Aceitação

A ferramenta foi apresentada à fábrica na presença dos gerentes de demanda e obteve uma boa recepção. A possibilidade de filtrar classes pelo conteúdo, nomenclatura e métricas possibilitou que questões antigas, como a cobertura em DAOs e DTOs, fossem resolvidas, ao menos do ponto de vista técnico. Ressalvas foram feitas por parte da fábrica de software no sentido de não utilizar a ferramenta como última instância no processo de

aceite, pois, dadas as peculiaridades de cada demanda, não seria possível, em certos casos, seguir um conjunto de métricas e limites globais. O *plugin* conta com a possibilidade de customizar a demanda por projetos de forma a possibilitar tal customização.

O eQualidade foi disponibilizado em um *dashboard* próprio na plataforma SonarQube, de forma que apenas os desenvolvedores interessados pudessem visualizá-lo no painel principal do Sonar. As métricas previstas no contrato foram incluídas no painel global da ferramenta, tornando-se, assim, disponíveis para todos os projetos.

Capítulo 5

Considerações Finais

Dados os objetivos preliminares traçados ao início deste trabalho, os resultados foram os seguintes:

1) A revisão da literatura levou a um conjunto de métricas orientadas a objetos, as quais tiveram sua significância e eficácia comprovadas empiricamente. Todavia, a aplicação desse conjunto de métricas passa por fatores políticos e limitações tecnológicas no processo do contrato, por exemplo: algumas métricas da suíte de Chidamber-Kemerer foram desligadas do plugin devido a pedidos da fábrica, outras foram depreciadas pela plataforma SonarQube ao longo do desenvolvimento da ferramenta [15].

2) O estudo do processo de controle de qualidade de software do TCU concluiu que os processos e técnicas empregados são vitais para o bom funcionamento do contrato com a fábrica de software. Órgãos públicos que pretendem realizar um contrato similar se beneficiariam muito das técnicas e processos descritos no capítulo 3.

3) Num esforço para escutar as sugestões e críticas dos usuários, foi aplicado um questionário entre os membros da fábrica e os gerentes de projeto do TCU. A aplicação do questionário não obteve significância estatística entre os membros da fábrica, mas o feedback dos envolvidos no contrato foi recebido em diversas reuniões sobre o contrato e a aplicação da ferramenta.

4) O plugin entrou em produção no final de 2014 e permanece até a data de publicação deste trabalho como parte do processo de aceite das órdens de serviço contratadas.

É, portanto, válido concluir que o projeto alcançou o objetivo de viabilizar o uso da plataforma sonarqube no contrato, solucionando o problema da dificuldade em avaliar os artefatos no processo de aceite. Houve ressalvas por parte da fábrica de software, no sentido de adaptar os limites das métricas a cada ordem de serviço, devido à grande variedade das entregas, de forma que o relatório das métricas não se tornasse a palavra final no processo de aceite. A ferramenta suporta essa necessidade através do painel de configuração por projeto e as adaptações a serem feitas em cada ordem de serviço são discutidas em reunião entre o TCU e a fábrica.

5.1 Trabalhos Futuros

O eQualidade cobre um aspecto da visualização de métricas que é pouco explorado pela plataforma SonarQube, e, portanto, conta com diversas oportunidades para expansão em trabalhos futuros. A internacionalização do *plugin* é essencial para divulgá-lo aos usuários do Sonar através da página de *plugins* da comunidade, e será o primeiro passo na evolução do *plugin*. O desenvolvimento da ferramenta ficou dividido entre o Serviço de Padronização e Arquitetura de Software (SEPAS) do TCU, que conta com uma cópia do *plugin* customizada para as necessidades da fábrica, e a versão descrita neste trabalho, que generaliza as funcionalidades desenvolvidas no TCU para melhor servir a um número maior de usuários. As versões continuarão a evoluir seguindo requisitos diferentes. O objetivo principal para os trabalhos futuros é ampliar a utilidade do *plugin*, possibilitando a criação de relatórios largamente customizáveis.

Referências

[1] Fernando Brito Abreu and Rogério Carapuça. Object-oriented software engineering: Measuring and controlling the development process. In *proceedings of the 4th International Conference on Software Quality*, volume 186, 1994. 11

[2] B. W. Boehm, J. R. Brown, and M. Lipow. Quantitative evaluation of software quality. In *Proceedings of the 2Nd International Conference on Software Engineering*, ICSE '76, pages 592–605, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press. vi, 6, 7, 9

[3] Nanette Brown, Yuanfang Cai, Yuepu Guo, Rick Kazman, Miryung Kim, Philippe Kruchten, Erin Lim, Alan MacCormack, Robert Nord, Ipek Ozkaya, et al. Managing technical debt in software-reliant systems. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pages 47–52. ACM, 2010. 15

[4] G Campbell and Patroklos P Papapetrou. *SonarQube in Action*. Manning Publications Co., 2013. 21

[5] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.*, 20(6):476–493, June 1994. 9

[6] Don Coleman, Dan Ash, Bruce Lowther, and Paul Oman. Using metrics to evaluate software system maintainability. *Computer*, 27(8):44–49, 1994. 8

[7] Vittorio Cortellessa, Harshinder Singh, and Bojan Cukic. Early reliability assessment of uml based software models. In *Proceedings of the 3rd International Workshop on Software and Performance*, WOSP '02, pages 302–309, New York, NY, USA, 2002. ACM. 8

[8] Tribunal de Contas da União. Relatório de levantamento. avaliação da governança de tecnologia da informação na administração pública federal. Technical report, Ministério do Planejamento, Orçamento e Gestão, 2014. 1

[9] Fabiane Giusti. Cast conquista quatro novos contratos no setor público. http://www.tce.ms.gov.br/portal/admin/db/clipping/4714.pdf, 09 2013. 1

[10] Miguel Griffa. Welcome to pmd, 2014. 20

[11] Tibor Gyimothy, Rudolf Ferenc, and Istvan Siket. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Trans. Softw. Eng.*, 31(10):897–910, October 2005. 16

[12] Jim Highsmith. *Agile Software Development Ecosystems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002. 5

[13] ISO. *ISO 9001:2008 Quality management systems - Requirements*, 2008. 5

[14] Robert JK Jacob and Keith S Karn. Eye tracking in human-computer interaction and usability research: Ready to deliver the promises. *Mind*, 2(3):4, 2003. 8

[15] Fabrice Bellingard Julien Lancelot. Deprecate rfc metric. https://jira.codehaus.org/browse/SONAR-5042, 04 2014. 30

[16] Stephen H Kan. *Metrics and models in software quality engineering*. Addison-Wesley Longman Publishing Co., Inc., 2002. 5

[17] Kohsuke Kawaguchi. Meet jenkins, 2015. 20

[18] Barbara Kitchenham and Shari Lawrence Pfleeger. Software quality: The elusive target. *IEEE software*, 13(1):12–21, 1996. 5

[19] Meir M Lehman. Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, 68(9):1060–1076, 1980. 1

[20] Bo Leuf and Ward Cunningham. The wiki way: collaboration and sharing on the internet. 2001. 21

[21] Wei Li and Sallie Henry. Object-oriented metrics that predict maintainability. *Journal of systems and software*, 23(2):111–122, 1993. 8

[22] Freddy Mallet. Fight back design erosion by breaking cycles with sonar, 2010. 14

[23] Thomas J McCabe. A complexity measure. *Software Engineering, IEEE Transactions on*, (4):308–320, 1976. 14

[24] Paulo Merson. Ultimate architecture enforcement: custom checks enforced at code-commit time. In *Proceedings of the 2013 companion publication for conference on Systems, programming, & applications: software for humanity*, pages 153–160. ACM, 2013. 20

[25] Sirsendu Mohanta, Gopika Vinod, A. K. Ghosh, and Rajib Mall. An approach for early prediction of software reliability. *SIGSOFT Softw. Eng. Notes*, 35(6):1–9, nov 2010. 8

[26] Wellton Máximo. Ministério do desenvolvimento estuda fazer censo para medir carência de mão de obra qualificada. http://memoria.ebc.com.br/agenciabrasil/noticia/2011-01-30/ministerio-do-desenvolvimento-estuda-fazer-censo-para-medir-carencia-de-mao-de-obra-qualificada, 01 2011. 1

[27] ABNT NBR. 9241-11. requisitos ergonômicos para trabalho de escritório com computadores: Parte 11—orientação sobre usabilidade. *ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. Rio de Janeiro: sn*, page 21, 2002. 7

[28] Jakob Nielsen. Why you only need to test with 5 users, 2000. 8

[29] Hector M Olague, Letha H Etzkorn, Sampson Gholston, and Stephen Quattlebaum. Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes. *Software Engineering, IEEE Transactions on*, 33(6):402–419, 2007. 15

[30] Paul Oman and Jack Hagemeister. Metrics for assessing a software system's maintainability. In *Software Maintenance, 1992. Proceerdings., Conference on*, pages 337–344. IEEE, 1992. 8

[31] José Romero, Oscar Pastor, and Jorge Belenguer. Methodological approach to software quality assurance through high-level object-oriented metrics. In *Proceedings of the 8th International Conference on Object-Oriented. Information Systems*, OOIS '02, pages 397–408, London, UK, UK, 2002. Springer-Verlag. 16

[32] Danilo Scarlet. Guia de aquisição, 2013. 1

[33] Orçamento e Gestão MP/SLTI Secretaria de Logística e Tecnologia da Informação do Ministério do Planejamento. Instrução normativa nº 4, de 12 de novembro de 2010. http://www.governoeletronico.gov.br/sisp-conteudo/nucleo-de-contratacoes-de-ti/modelo-de-contratacoes-normativos-e-documentos-de-referencia/instrucao-normativa-mp-slti-no04, 11 2010. 1

[34] Yogesh Singh, Arvinder Kaur, and Ruchika Malhotra. Empirical validation of object-oriented metrics for predicting fault proneness models. *Software Quality Control*, 18(1):3–35, March 2010. 16

[35] Sherif M. Yacoub, Bojan Cukic, and Hany H. Ammar. Scenario-based reliability analysis of component-based software. In *Proceedings of the 10th International Symposium on Software Reliability Engineering*, ISSRE '99, pages 22–, Washington, DC, USA, 1999. IEEE Computer Society. 8

[36] Yuming Zhou and Hareton Leung. Predicting object-oriented software maintainability using multivariate adaptive regression splines. *Journal of Systems and Software*, 80(8):1349–1361, 2007. 8

[37] Hong Zhu, Patrick AV Hall, and John HR May. Software unit test coverage and adequacy. *Acm computing surveys (csur)*, 29(4):366–427, 1997. 14